Blaze

Jan-Peter Krämer, Joachim Kurz, Thorsten Karrer, Jan Borchers
RWTH Aachen University
52062 Aachen, Germany
{*kraemer, kurz, karrer, borchers*}*@cs.rwth-aachen.de*

***Abstract*—Understanding source code is crucial for successful software maintenance. To understand source code, navigation in the call graph has been shown to be particularly important. Programmers often employ a two-phased strategy for effective call graph exploration. We present *Blaze,* a source code exploration tool designed to explicitly support this strategy. In a study, we show that call graph exploration tools significantly increase success rates in typical software maintenance tasks and that using Blaze significantly reduces task completion times compared to using the Call Hierarchy or Xcode.**

***Keywords*-Tools and environments, Software visualization, Program comprehension**

I. Introduction

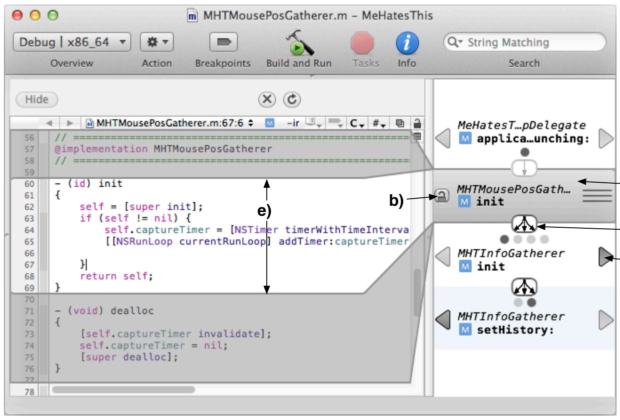
Software maintenance, the task of fixing bugs, adding new features, and performing other modifications after software has been shipped, accounts for up to 70% of the total expenses in a software project [4]. Successful software maintenance requires finding the correct location for a change in the source code and performing the change without introducing side effects. To do so, the developer has to thoroughly understand the code. While numerous tools have been proposed to support this task, software developers still consider comprehending unknown code one of their biggest problems [5].

A key activity in software maintenance is to navigate source code [6]. Among the various types of navigation, *call graph based navigation* is especially important to determine where to implement a change [1], [7], [8]. The call graph contains all methods as nodes, and each method is connected to its callees through outgoing edges and to its callers through incoming edges.

Previous studies [2], [3], [7] document a *two-phase model* for navigation in the call graph: Developers start by searching for an *anchor point* they consider interesting (Phase 1). Once they have found an anchor point, they follow different paths starting from there until they reach the part of the code they need to work with (Phase 2). This navigation behavior was observed in multiple, independent studies with different settings. *Blaze* (Fig. 1), was designed to offer support for this two-phase model.

II. Related Work

Many current IDEs, for example Eclipse or Visual Studio, implement a Call Hierarchy tool that uses a tree view to



```
// ================================================================
@implementation MHTMousePosGatherer
// ================================================================

- (id) init
{
    self = [super init];
    if (self != nil) {
        self.captureTimer = [NSTimer timerWithTimeInterva
        [[NSRunLoop currentRunLoop] addTimer:captureTimer

    }
    return self;
}

- (void) dealloc
{
    [self.captureTimer invalidate];
    self.captureTimer = nil;
    [super dealloc];
}
```

Figure 1. Blaze visualizes a complete path in the call graph that includes the currently edited method. The developer can navigate up- and downstream from the focus method along the path, and change the path by selecting alternative entries using the arrow buttons next to each method.

show multiple levels of callers or callees, starting from a given focus method as the root of the tree view. Users change the root node explicitly by selecting a method identifier in the source code as the new focus method. Call Hierarchy tools are primarily useful during Phase 2 [10].

Stacksplorer [7] provides support for Phase 1. It gives users access to the call graph neighborhood of a focus method, i.e., the method the user is currently working on. To show this information, two interactive views are added as side columns next to the source code editor. While the central source code editor shows the focus method in the usual way, the left side column shows a list of callers, and the right side column a list of callees. Clicking on any method in the side columns navigates to this method. Whenever the focus method changes in the central editor, the side columns update automatically. Optional graphical overlays connecting method calls in the source code with the corresponding entries in the side columns help making sense of the displayed information.

III. Blaze

Blaze is designed to support both phases of the two-phase navigation model. It adds an interactive view on the right side of the source code editor (Fig. 1). This view shows a single path through the call graph containing the focus method, which is displayed in gray (a)). During Phase 1, Blaze is *unlocked* and the focus method updates automatically to match the method the user is currently working on. When the method currently being edited changes, the focus method changes accordingly, and the displayed path is updated. An optional overlay connects the currently edited method in the editor to the corresponding entry in the side column (e)). The additional information scent that Blaze provides while it is unlocked is important during Phase 1.

During Phase 2, Blaze can be *locked* (b)) to prevent the focus method and the path displayed in Blaze from changing due to navigation in the editor. The focus method becomes the current anchor point. Blaze now allows developers to more easily explore all paths starting from this anchor point.

For each entry on the displayed path, several alternatives may exist. Because the focus method has to remain on the path, each entry below the focus method can be exchanged with another callee of the preceding method; above the focus method, each entry can be exchanged with another caller of the following method. When an entry is exchanged, the following (below the focus method) or preceding (above the focus method) path changes accordingly. To exchange an entry on the path, a user can either click the arrows between two entries (c)) to reveal a menu with all options, or use the arrows next to each entry (d)) to flip through the alternatives. This interface contains as much information as the Call Hierarchy tool, but consumes relatively little screen space because it never shows more than one single path at a time. It also allows for a very structured (depth-first) exploration of all possible paths that include the focus method.

We developed a fully functional prototype of Blaze as a plug-in for Apple's Xcode IDE. It is developed using Objective-C and utilizes Xcode's internal source code parsers.

IV. Study

We studied developers working on maintenance tasks using Blaze, a Call Hierarchy tool, Stacksplorer, or an unmodified Xcode IDE. 35 students participated in the between groups study, two of which were removed from the evaluation due to technical problems, yielding 9 participants in the Call Hierarchy condition and 8 participants in all remaining conditions. The participants had to solve two tasks in 40 minutes. If the participants proposed an incorrect solution for a task or time ran out, we considered the participant not successful.

All call graph exploration tools led to a significantly increased number of successful participants compared to Xcode alone. Both Stacksplorer and Blaze could reduce task completion times significantly compared to the status quo, i.e., the unmodified Xcode installation and the Call Hierarchy. Surprisingly, Blaze could not outperform Stacksplorer in terms of task completion time, although Blaze should, by design, more completely support the two-phase navigation model than Stacksplorer. One possible explanation for this result is that Stacksplorer's support for Phase 1 is superior to Blaze's, so Stacksplorer can compensate for its missing specific support for Phase 2.

V. Future Work

We plan to study in more detail why Blaze was no significant improvement over Stacksplorer. For that, we will analyze the actual navigation paths from our video recordings of the study sessions to see how they were influenced by the tools used.

Acknowledgements

This work was funded in part by the German B-IT Foundation and by the German Government through its UMIC Excellence Cluster at RWTH Aachen University.

References

[1] T. D. LaToza and B. A. Myers, "Developers Ask Reachability Questions," in *Proc. ICSE '10*. ACM, 2010.

[2] A. Ko, B. Myers, M. Coblenz, and H. Aung, "An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, 2006.

[3] J. Sillito, G. C. Murphy, and K. D. Volder, "Asking and Answering Questions during a Programming Change Task," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, 2008.

[4] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. McGraw-Hill, 2010.

[5] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining Mental Models: A Study of Developer Work Habits," in *Proc. ICSE '06*. ACM, 2006.

[6] M. P. Robillard, W. Coelho, and G. C. Murphy, "How Effective Developers Investigate Source Code:An Exploratory Study," *IEEE Trans. Softw. Eng.*, vol. 30, no. 12, 2004.

[7] T. Karrer, J.-P. Krämer, J. Diehl, B. Hartmann, and J. Borchers, "Stacksplorer: Call graph navigation helps increasing code maintenance efficiency," in *Proc. UIST '11*. ACM, 2011.

[8] T. Winograd, "Breaking the complexity barrier again," *ACM SIGIR Forum*, 1974.

[9] J. Lawrance, R. Bellamy, M. Burnett, and K. Rector, "Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks," in *Proc. CHI '08*. ACM, 2008.

[10] J.-P. Krämer, J. Kurz, T. Karrer, and J. Borchers, "Blaze: Supporting two-phased call graph navigation in source code," in *Proc. CHI '12*, ser. CHI EA '12. New York, NY, USA: ACM, 2012.